Binary Exponentiation in Java (x■ Computation)

Concept

Binary exponentiation is an efficient method to compute x■ (x raised to the power n) in O(log n) time, compared to the naive approach that takes O(n) time.

Naive Approach (Linear)

```
double result = 1;
for(int i = 0; i < n; i++) {
    result *= x;
}
```

Complexity: O(n)

Binary Exponentiation (Efficient Approach)

Idea:

1. Express n in binary form.
2. Multiply x only when the corresponding bit is 1.
3. Square x at each step.
4. Reduce n by half at each iteration.

Complexity:
- Time: O(log n)
- Space: O(1)

Pseudocode

```
function power(x, n):
    ans = 1
    while n > 0:
        if n is odd:
            ans = ans * x
        x = x * x
        n = n / 2
    return ans
```

Java Implementation

```
class Solution {
    public double myPow(double x, int n) {
        long N = n; // handle negative powers safely
        if(N < 0) {
            x = 1 / x;
            N = -N;
        }

        double ans = 1;
        while(N > 0) {
            if(N % 2 == 1) { // if the least significant bit is 1
                ans *= x;
            }
            x *= x; // square x
```

```
            N /= 2; // shift right (divide by 2)
        }

        return ans;
    }
}
```

Notes:

- Using long N ensures we handle Integer.MIN_VALUE correctly.
- Works for both positive and negative powers.
- Highly efficient for very large n.